PR #35451 完整报告

sgl-project/sglang

[Feature] Support PP in full prefill CUDA graphs

合并时间：2026-08-28 08:32

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35451

执行摘要

- 一句话：支持流水线并行下的全量 prefill CUDA 图回放
- 推荐动作：值得精读。该 PR 展示了如何在 CUDA graph 捕获 / 回放机制中安全地引入 pipeline 输入：通过 buffer registry 的 GraphSlot + source_fn 模式解决图捕获所需的数据指针稳定性，用 _align_pipeline_layers 解决 PP 层索引对齐，设计清晰且可复用。关注 cuda_graph_buffer_registry.py 的 slot 注册模式与 prefill_cuda_graph_runner.py 的非首 rank 输入切换逻辑，可作为后续扩展多模态或 residual stream 场景的参考。

功能与动机

PR body 明确指出："Full prefill CUDA graphs currently cannot replay pipeline-parallel stages. The runner has no graph-stable storage for pipeline inputs, does not pass those inputs to non-first stages, and rejects pipeline proxy outputs." 即现有实现中 PrefillCudaGraphRunner 在遇到 PPProxyTensors 输出时会直接 raise NotImplementedError，导致 PP 模型无法享受 full prefill CUDA graph 的加速收益。

实现拆解

1. 分配 PP proxy 缓冲（buffers.py）：在 PrefillInputBuffers.create 中新增 pp_size、hc_hidden_size、pp_proxy_topk_size、pp_proxy_residual_num_blocks 参数。当 pp_size > 1 时，按 token-major 布局分配 pp_proxy_tensors（包含 hidden_states，以及非 MHC 模型下的 residual、可选 topk_indices），为 CUDA graph 捕获提供稳定的张量地址。
2. 注册 graph slot（cuda_graph_buffer_registry.py）：在 build_prefill_registry 末尾，将 source.pp_proxy_tensors 中的每个张量注册为 GraphSlot，axis="tokens"、padding_policy=ZERO，并通过 _pp_source(key) 生成的 source_fn 在回放时从 ForwardBatchContext.pp_proxy_tensors 读取实际数据，保证捕获与回放的数据指针一致。
3. 对齐全局层索引（cuda_graph_setup.py）：新增 _align_pipeline_layers 工具函数，依据 layer_model.start_layer / end_layer 将局部 attention_layers 与 mha_companion_layers 用 None 占位补回全局位置，避免 PP 切分导致后续层遍历下标错位。
4. 接通 runner 路径（prefill_cuda_graph_runner.py）：构造静态 _static_pp_proxy_tensors 包装；在 _run_forward 中对非首个 PP rank 传入 None 的 input_ids / input_embeds 并注入 pp_proxy_tensors 到 layer_model.forward；

load_batch 透传 pp_proxy_tensors；_finalize_execute_output 移除 NotImplementedError 并裁剪输出到 raw_num_tokens。

5. 测试配套：单元测试覆盖 PP proxy 缓冲分配形状、_finalize_execute_output 对 PPProxyTensors 的裁剪、_align_pipeline_layers 的全局索引对齐与非法参数断言；双 GPU e2e 测试 TestFullCudaGraphPipelineParallel::test_pp_replays_full_prefill_cuda_graph 验证 47-token prompt 走 64-token prefill bucket 且 cuda graph: True。

关键文件：

- python/sglang/srt/model_executor/cuda_graph_buffer_registry.py（模块 缓冲注册；类别 source；类型 data-contract；符号 _pp_source, _fn）：在 build_prefill_registry 中将 PP proxy 张量注册为 GraphSlot，是 CUDA graph 捕获 / 回放稳定性的核心数据契约变更。
- python/sglang/srt/model_executor/model_runner_components/cuda_graph_setup.py（模块 图捕获设置；类别 source；类型 data-contract；符号 _align_pipeline_layers）：新增 _align_pipeline_layers 并在捕获前对 attention_layers / mha_companion_layers 做全局索引对齐，是 PP 下捕获正确性的关键。
- python/sglang/srt/model_executor/runner/prefill_cuda_graph_runner.py（模块 预填图执行；类别 source；类型 data-contract；符号 _run_forward, _finalize_execute_output, load_batch）：runner 主路径：构造静态 PP proxy 缓冲、非首 rank 输入切换、load_batch 透传、放开 PPProxyTensors 输出裁剪，是功能生效的核心。
- python/sglang/srt/model_executor/runner_utils/buffers.py（模块 输入缓冲；类别 source；类型 data-contract；符号 PrefillInputBuffers.create）：PrefillInputBuffers 新增 pp_proxy_tensors 字段与分配逻辑，是 graph-stable 存储的物理载体。
- test/registered/unit/model_executor/test_prefill_cuda_graph_runner_helpers.py（模块 单元测试；类别 test；类型 test-coverage；符号 test_prefill_buffers_allocate_pipeline_proxy_token_rows, test_pipeline_proxy_output_is_supported）：覆盖 PP proxy 缓冲分配形状与 PPProxyTensors 输出裁剪，验证新契约的正确性。
- test/registered/cuda_graph/full_prefill/test_full_cuda_graph_prefill.py（模块 端到端测试；类别 test；类型 test-coverage；符号 TestFullCudaGraphPipelineParallel, test_pp_replays_full_prefill_cuda_graph）：新增双 GPU e2e 测试，验证 PP 下 full prefill CUDA graph 真实回放并断言请求确实走了 prefill graph。

关键符号：_align_pipeline_layers, _pp_source, _fn, PrefillInputBuffers.create, PrefillCudaGraphRunner._run_forward, PrefillCudaGraphRunner._finalize_execute_output, PrefillCudaGraphRunner.load_batch

关键源码片段

python/sglang/srt/model_executor/cuda_graph_buffer_registry.py

在 build_prefill_registry 中将 PP proxy 张量注册为 GraphSlot，是 CUDA graph 捕获 / 回放稳定性的核心数据契约变更。

```
# python/sglang/srt/model_executor/cuda_graph_buffer_registry.py
# 在 build_prefill_registry 末尾追加：若 source 上存在 PP 代理张量，
# 则把它们注册为 graph slot，使 CUDA graph 捕获 / 回放期间拥有稳定存储。
```

```
    if source is not None:
        pp = getattr(source, "pp_proxy_tensors", None)
        if pp is not None:

            # 为每个 key 生成 source_fn，回放时从 ForwardBatchContext 中
            # 动态取出实际张量，避免闭包捕获可变 key 造成脏数据。
            def _pp_source(key):
                def _fn(_fb, ctx):
                    ppx = ctx.pp_proxy_tensors
                    return None if ppx is None else ppx.tensors[key]

                return _fn

            for _key, _backing in pp.items():
                reg.register_slot(
                    GraphSlot(
                        name=f"pp_proxy_tensors.{_key}",
                        shape_fn=lambda _bs, _mt, _s=tuple(_backing.shape): _s,
                        dtype=_backing.dtype,
                        axis="tokens",
                        padding_policy=PaddingPolicy.ZERO,
                        source_fn=_pp_source(_key),
                    ),
                    bind=_backing,
                )
    return reg
```

python/sglang/srt/model_executor/model_runner_components/cuda_graph_setup.py

新增 _align_pipeline_layers 并在捕获前对 attention_layers / mha_companion_layers 做全局索引对齐，是 PP 下捕获正确性的关键。

```
# python/sglang/srt/model_executor/model_runner_components/cuda_graph_setup.py
def _align_pipeline_layers(layers: list, layer_model) -> list:
    # 当模型被流水线并行切分时，start_layer / end_layer 标记了本 stage
    # 负责的层区间，而 layers 列表只包含局部层；为保证后续按全局索引
    # 遍历时下标一致，这里用 None 占位补齐前后缀。
    has_start_layer = hasattr(layer_model, "start_layer")
    has_end_layer = hasattr(layer_model, "end_layer")
    assert (
        has_start_layer == has_end_layer
    ), "pipeline layer ranges must define start_layer and end_layer together"

    start_layer = layer_model.start_layer if has_start_layer else 0
    end_layer = layer_model.end_layer if has_end_layer else len(layer_model.layers)
    assert isinstance(start_layer, int) and isinstance(
        end_layer, int
    ), "pipeline layer ranges must define integer start_layer and end_layer"
    assert 0 <= start_layer <= end_layer <= len(layer_model.layers), (
```

```
        f"invalid pipeline layer range [{start_layer}, {end_layer}) for "
        f"{len(layer_model.layers)} layers"
    )
    assert (
        len(layers) <= end_layer - start_layer
    ), f"found {len(layers)} layers in PP range [{start_layer}, {end_layer})"

    return (
        [None] * start_layer + layers + [None] * (len(layer_model.layers) - end_layer)
    )
```

python/sglang/srt/model_executor/runner/prefill_cuda_graph_runner.py

runner 主路径：构造静态 PP proxy 缓冲、非首 rank 输入切换、load_batch 透传、放开 PPProxyTensors 输出裁剪，是功能生效的核心。

```
# python/sglang/srt/model_executor/runner/prefill_cuda_graph_runner.py
# Full CG 后端下，非首个 PP stage 不消费 input_ids / input_embeds，
# 而是从静态 PP proxy 张量读取上游传递的 hidden_states 等数据。
if self._uses_eager_prefill_tail():
    # BCG / Full：只捕获 transformer body。
    positions = self._get_layer_model_positions(forward_batch)
    input_ids = forward_batch.input_ids
    input_embeds = forward_batch.input_embeds
    layer_kwargs = {}
    if self._static_pp_proxy_tensors is not None:
        layer_kwargs["pp_proxy_tensors"] = self._static_pp_proxy_tensors[:num_tokens]
    if not self.model_runner.pp_group.is_first_rank:
        input_ids = None
        input_embeds = None
    return self.layer_model.forward(
        input_ids,
        positions,
        forward_batch,
        input_embeds,
        **layer_kwargs,
    )
```

评论区精华

Oasis-Git 在 review 中总体肯定实现（"Overall the implementation is good"），并提出两点：一是询问 mock model 的 PP *e2e* 测试中为什么要禁用 *decode* CUDA graph（review 评论于 test_e2e_pp.py 的 diff），二是建议在 test/registered/cuda_graph/full_prefill/ 下补充专门的 *e2e* 测试。作者在 issue 评论中回复 "added!" 并实际新增了 test_full_cuda_graph_prefill.py；对于 *decode* graph 禁用问题，评论区未见直接答复，但 PR 最终通过合并，推断该测试改动是为了隔离验证 prefill graph 路径。

- 测试中为何禁用 *decode* CUDA graph (question): 评论区内未见直接答复；结合 PR body，该测试改动是为隔离验证 full prefill CUDA graph 路径，PR 最终合并。

- 是否补充 full_prefill *e2e* 测试 (testing): 作者回复 "added!"，并实际新增了 test_full_cuda_graph_prefill.py 中的双 GPU *e2e* 测试。
- 整体实现评价 (design): 设计得到认可，仅对测试覆盖提出补充要求，作者完成后通过。

风险与影响

- 风险：
  1. 非首 rank 输入置空风险：prefill_cuda_graph_runner.py 的 _run_forward 在非首个 PP stage 将 input_ids / input_embeds 置为 None，若后续某些模型实现仍会在非首 stage 读取这两个参数，可能引发意外错误，需要模型侧契约保证。
  2. 层索引对齐依赖断言：_align_pipeline_layers 使用 assert 校验 start_layer / end_layer 的合法性与配对性，一旦模型配置不合规会直接中断捕获流程；虽然是显式失败，但错误信息依赖模型实现细节。
  3. PP proxy 缓冲容量假设：pp_proxy_tensors 按 max_num_tokens 分配，回放时依赖 ctx.pp_proxy_tensors 存在且形状匹配，若上游未正确传递或形状超界，source_fn 返回 None 或越界截断可能导致静默错误。
  4. 测试覆盖有限：功能仅通过双 GPU mock 模型验证，真实模型（如带 residual stream 的架构）下的行为未被覆盖，存在一定的回归风险。 - 影响：对用户而言，启用 pipeline parallelism 的部署现在可以享受 full prefill CUDA graph 的加速能力，prefill 延迟预期下降；对系统而言，改动集中在 CUDA graph 捕获与回放的基础设施，但不触碰任何数值内核，非 PP、非 prefill-graph 路径完全不受影响。对团队而言，该 PR 为后续 PP + FullCG 的组合排除了一个明确的 NotImplementedError 障碍，降低了 PP 场景下的性能调优门槛。影响范围中等，主要受益者是使用 PP 且开启 full prefill CUDA graph 的用户。 - 风险标记：PP 新路径测试覆盖有限，非首 rank 输入置空依赖模型契约，层索引对齐断言可能中断捕获，PP proxy 缓冲形状依赖上游

关联脉络

- 暂无明显关联 PR